

Nosso primeiro teste unitário com javascript

Índice

- 01** Configurando o framework
- 02** Nosso primeiro teste
- 03** Agrupando teste unitário



01

Configurando o framework

Framework para Javascript

Para JavaScript, vamos usar o framework **Jest**.
Se quisermos configurá-lo, devemos instalar:

1. Um IDE — o recomendado é o Visual Studio Code (<https://code.visualstudio.com>).
2. Node.js (<https://nodejs.org>).

Devemos ter em mente que, se o código fornecido não tiver o arquivo **package.json**, devemos criá-lo, porque todas as configurações do nosso projeto serão salvas aqui.

Para criá-lo, devemos executar o comando no terminal: **npm init -y**



Framework para Javascript

3. JEST (<https://jestjs.io/>)

Para incluir o Jest em nosso projeto, devemos executar o comando no terminal: **npm install --save-dev jest** (lembre-se de que devemos fazer isso em cada um dos projetos em que vamos trabalhar).



02

Nosso primeiro teste

Criando nosso primeiro teste unitário

Baixe o código-fonte do seguinte repositório:

<https://github.com/academind/js-testing-introduction/tree/starting-setup>.

Este programa solicita a entrada de **Nome** e **Idade** e ao pressionar o botão Add User, ele compila uma lista com os dados inseridos.

Name

Tiago Gomes

Age

27

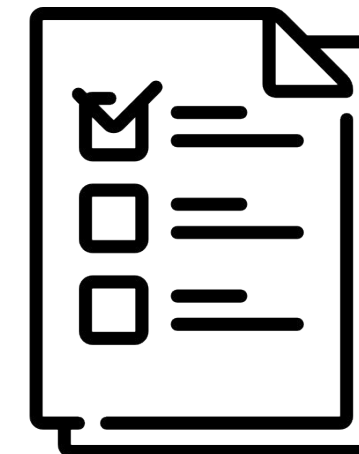
Add User

Tiago Gomes (27 years old)

Criando nosso primeiro teste unitário

Configure o **Jest** como executor de teste:

1. Vá para o arquivo **package.json**.
2. Na parte de **scripts/teste** devemos substituir “**echo “Error: no test specified” && exit 1**” por “**jest**”. Dessa forma, indicamos que vamos executar testes com nosso framework Jest.



The screenshot shows the Visual Studio Code interface with the following components:

- Menu Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Explorer Panel:** Displays the file structure of the project 'JS-TESTING-INTRODUCT...'. The files listed are:
 - __test__
 - util.test.js
 - .vscode
 - launch.json
 - dist
 - main.js
 - node_modules
 - .gitignore
 - app.js
 - index.html
 - package-lock.json
 - package.json** (selected)
 - styles.css
 - util.js
- Editor Panel:** Shows the content of the selected 'package.json' file. The code is as follows:


```

1  {
2    "name": "js-testing-introduction",
3    "version": "1.0.0",
4    "description": "An introduction to JS testing",
5    "main": "app.js",
6    "scripts": {
7      "start": "webpack app.js --mode development --watch",
8      "test": "jest"
9    },
10   "keywords": [
11     "js",
12     "javascript",
13     "testing",
14     "jest",
15     "unit",
16     "tests",
17     "integration",

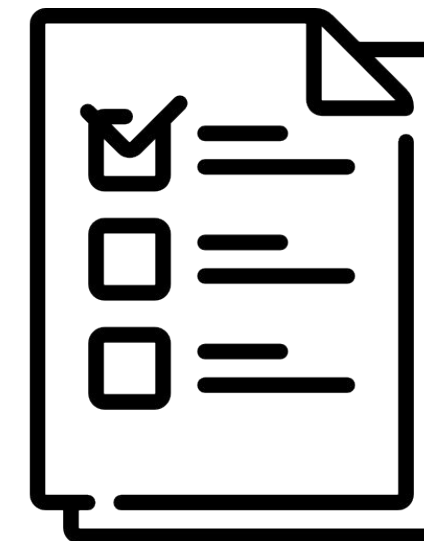
```

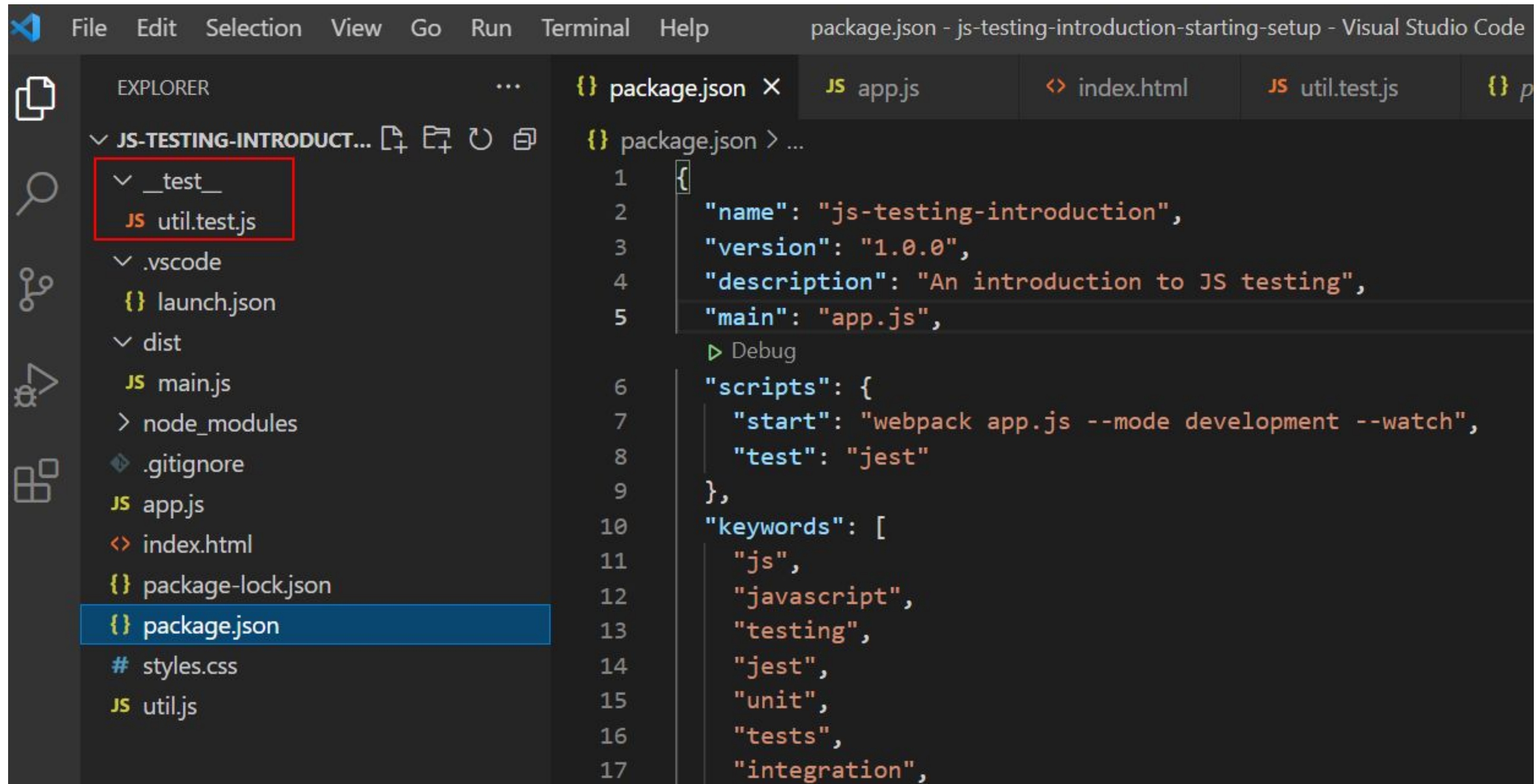
 The "test": "jest" line is highlighted with a red box.

Criando nosso primeiro teste unitário

Crie pasta e arquivo de teste:

1. Crie a pasta **__test__** onde todos os arquivos de teste estarão localizados.
2. Crie o arquivo de teste **FileName.test.js**. Para que o Jest reconheça os arquivos de teste, eles devem terminar em **.test.js**. Neste caso, nosso arquivo será chamado de **util.test.js**.





Visual Studio Code interface showing the Explorer and Editor views.

Explorer View (Left):

- JS-TESTING-INTRODUCT...
- __test_
 - util.test.js
- .vscode
 - launch.json
- dist
 - main.js
- node_modules
- .gitignore
- app.js
- index.html
- package-lock.json
- package.json** (Selected)
- styles.css
- util.js

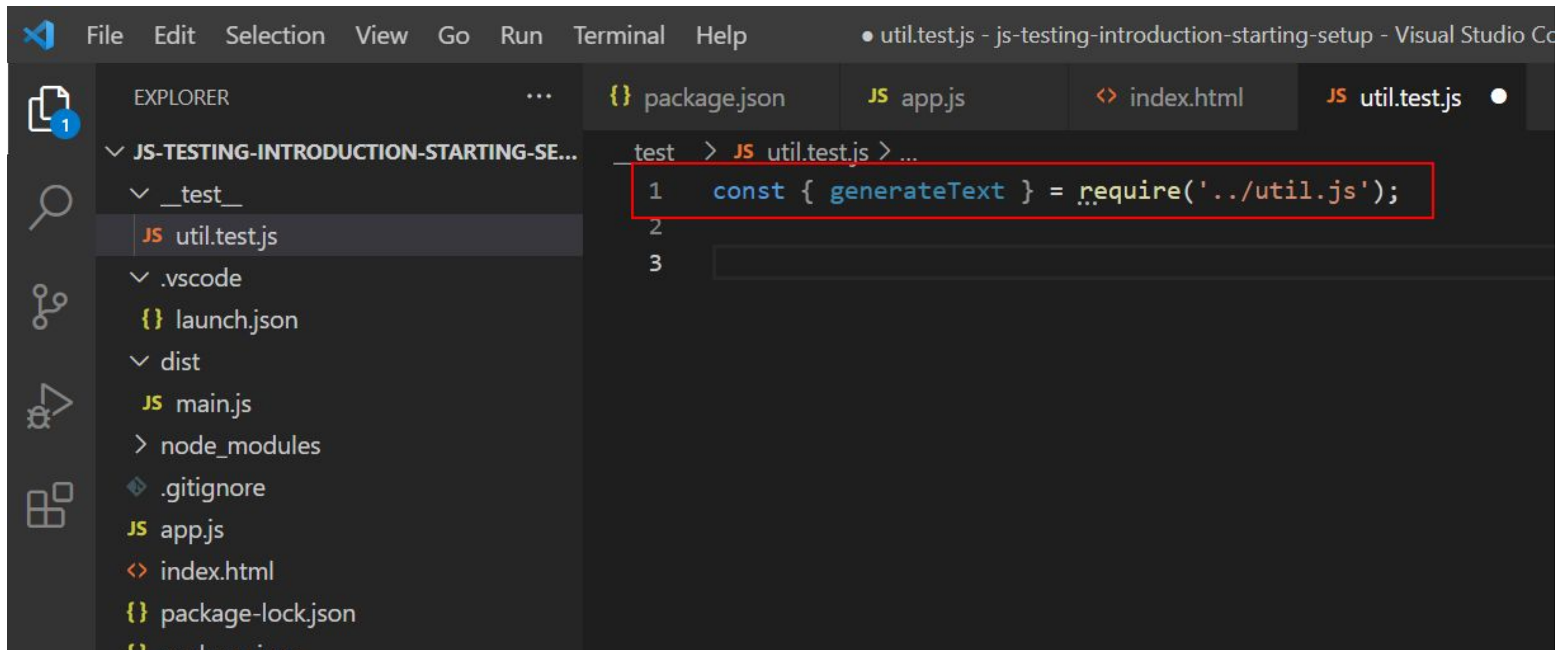
Editor View (Right):

```

package.json > ...
1  {
2    "name": "js-testing-introduction",
3    "version": "1.0.0",
4    "description": "An introduction to JS testing",
5    "main": "app.js",
6    "scripts": {
7      "start": "webpack app.js --mode development --watch",
8      "test": "jest"
9    },
10   "keywords": [
11     "js",
12     "javascript",
13     "testing",
14     "jest",
15     "unit",
16     "tests",
17     "integration",

```


Importar o código para testar: devemos importar a seção do código que queremos testar. Neste caso, vamos tentar o que se encontra no arquivo **util.js**, pois é onde está localizada a lógica do nosso programa.



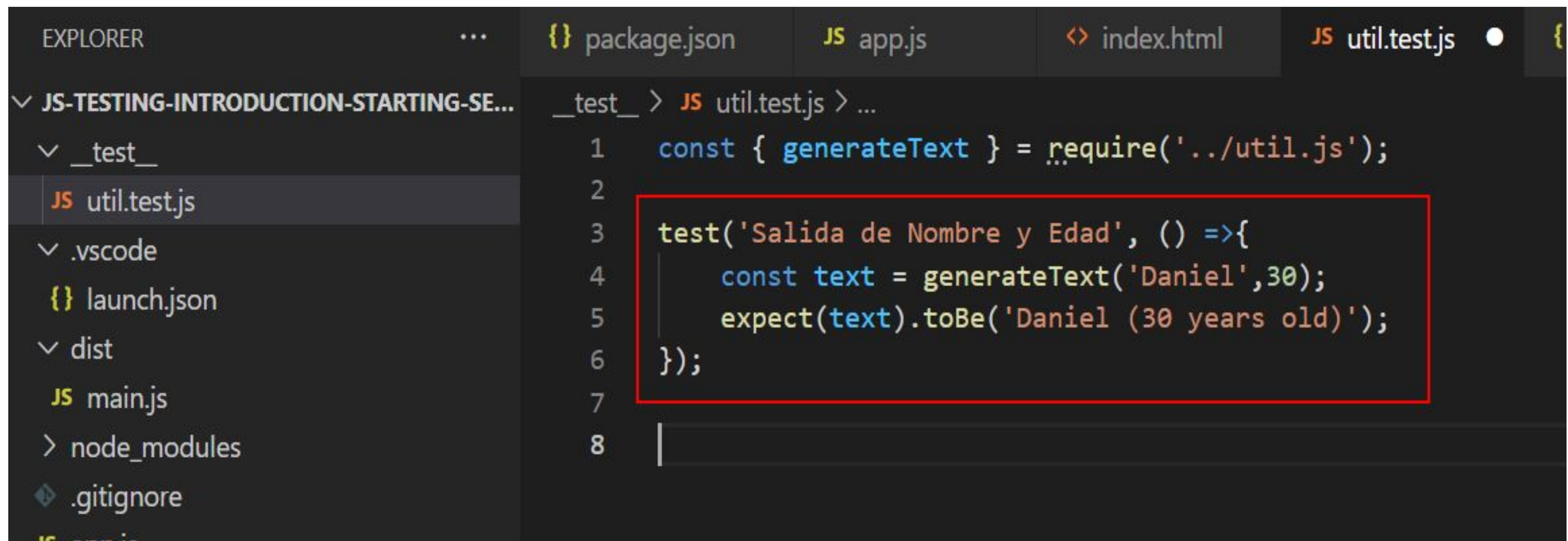
The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays the project structure for 'js-testing-introduction-starting-setup'. The file 'util.test.js' is selected under the '__test__' folder. The main editor area shows the content of 'util.test.js', with the first line of code highlighted by a red box:

```
1  const { generateText } = require('../util.js');
```

The breadcrumb at the top of the editor indicates the current file path: `test > JS util.test.js > ...`. The file explorer also lists other files like 'package.json', 'app.js', 'index.html', and 'package-lock.json'.

Escreva o código de teste de unidade. As principais funções a serem usadas são:

- **test()** ou **it()**: onde o teste é definido. Um nome de representante deve ser inserido.
- **expect()**: O que você deseja verificar faz parte da biblioteca de asserções. No exemplo, queremos verificar a saída gerada ao pressionar o botão Adicionar usuário.



The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the file structure of a project named 'JS-TESTING-INTRODUCTION-STARTING-SE...'. The file 'util.test.js' is selected under the '__test__' directory. The main editor window shows the code in 'util.test.js'. The code includes a require statement for 'generateText' and a unit test function. The test function is highlighted with a red rectangle. The test function is named 'Salida de Nombre y Edad' and takes an empty array as an argument. It calls 'generateText' with 'Daniel' and '30', and then uses 'expect' to verify that the returned text is 'Daniel (30 years old)'.

```
__test__ > JS util.test.js > ...
1  const { generateText } = require('../util.js');
2
3  test('Salida de Nombre y Edad', () =>{
4      const text = generateText('Daniel',30);
5      expect(text).toBe('Daniel (30 years old)');
6  });
7
8  |
```

Para executar o teste, devemos primeiro abrir o terminal e depois executar o comando `npm test`.

The screenshot shows the VS Code interface with the Explorer sidebar on the left displaying the project structure. The main editor shows the file `util.test.js` with the following code:

```
__test__ > JS util.test.js > test('Salida de Nombre y Edad') callback
1  const { generateText } = require('../util.js');
2
3  test('Salida de Nombre y Edad', () => {
4      const text = generateText('Daniel', 30);
5      expect(text).toBe('Daniel (30 years old)');
6  });
7
```

Below the editor, the TERMINAL tab is active, showing the command `npm test` and its output:

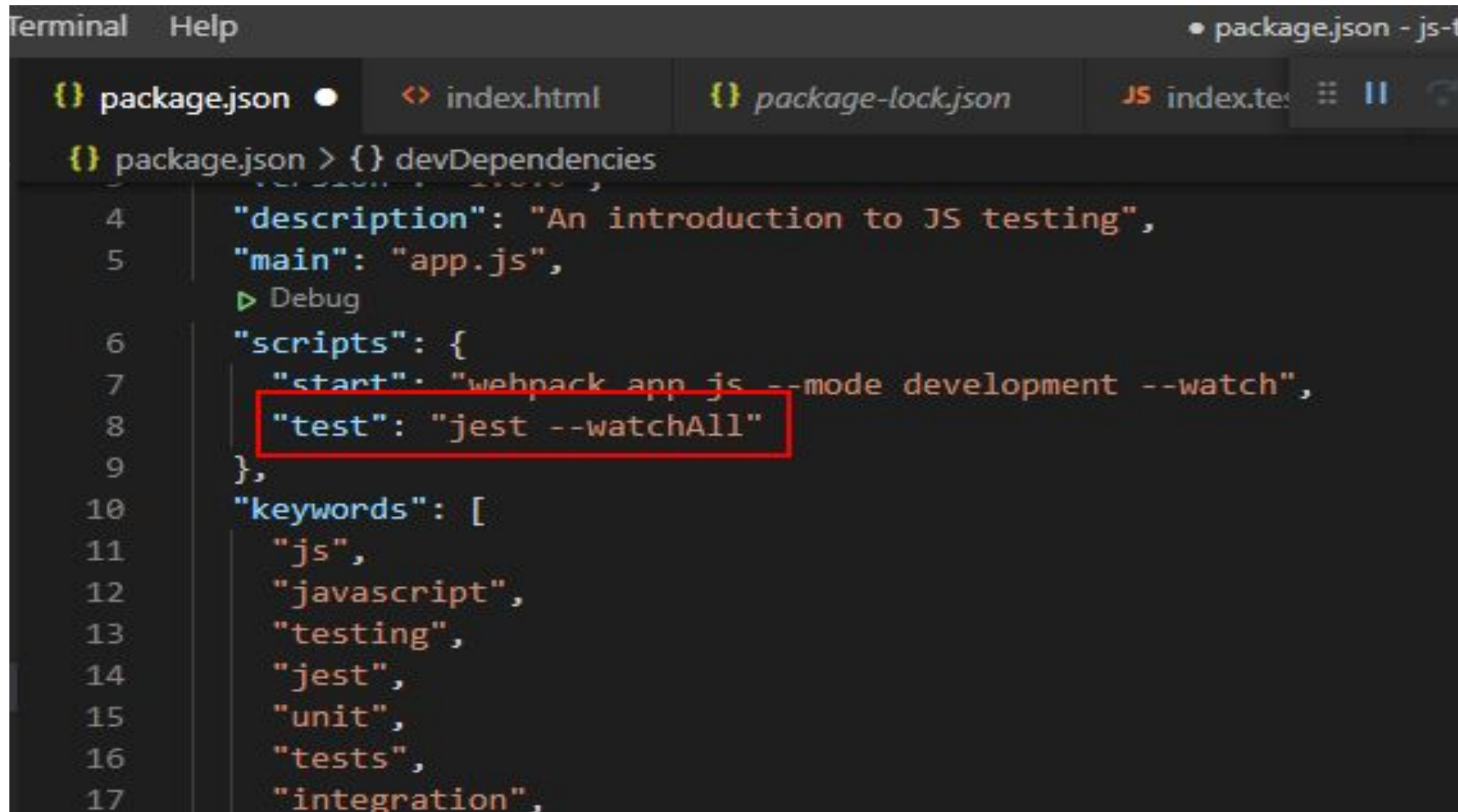
```
PS C:\unitTest\js-testing-introduction-starting-setup> npm test

> js-testing-introduction@1.0.0 test
> jest

PASS __test__/util.test.js
  ✓ Salida de Nombre y Edad (1 ms)
  ✓ Salida sin datos
  ✓ Sin datos

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.382 s, estimated 1 s
Ran all test suites.
```


Os testes podem ser configurados para serem executados sempre que uma alteração de código for gerada. Para isso, devemos ir ao arquivo package.json e digitar **jest --watchAll** no “test” que está dentro de “scripts”.

A screenshot of a code editor window showing the package.json file. The editor has a dark theme and a sidebar on the left with file explorer icons. The main area displays the JSON content of package.json. The 'scripts' object is expanded, and the 'test' property is highlighted with a red rectangle. The 'test' value is 'jest --watchAll'. Other visible properties include 'description', 'main', and 'keywords'.

```
Terminal  Help  • package.json - js-t  
{} package.json  {} index.html  {} package-lock.json  JS index.test.js  
{} package.json > {} devDependencies  
4  "description": "An introduction to JS testing",  
5  "main": "app.js",  
   ▶ Debug  
6  "scripts": {  
7    "start": "webpack app.js --mode development --watch",  
8    "test": "jest --watchAll"  
9  },  
10 "keywords": [  
11   "js",  
12   "javascript",  
13   "testing",  
14   "jest",  
15   "unit",  
16   "tests",  
17   "integration",
```

Em seguida, execute o seguinte comando:
npm test.

Lá, alguma modificação deve ser feita no código do programa (**util.js**) e, por fim, ao salvar o arquivo modificado, os testes serão executados.

EXPLORER

- JS-TESTING-INTRODUCTION-STARTING-SE...
- __test__
 - util.test.js
- .vscode
 - launch.json
- dist
 - main.js
- node_modules
- .gitignore
- app.js
- index.html
- package-lock.json
- package.json
- styles.css
- util.js

package.json app.js index.html util.test.js package-lock

```

JS util.js > generateText > generateText
1  exports.generateText = (name, age) => {
2    // Returns output text
3    return `${name} (${age} años)`;
4  };
5
6  exports.createElement = (type, text, className) => {
7    // Creates a new HTML element and returns it
8    const newElement = document.createElement(type);
9    newElement.classList.add(className);
10   newElement.textContent = text;

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

FAIL __test__/util.test.js

- × Salida de Nombre y Edad (1 ms)
- Salida de Nombre y Edad

```

expect(received).toBe(expected) // Object.is equality

Expected: "Daniel (30 years old)"
Received: "Daniel (30 años)"

3 | test('Salida de Nombre y Edad', () =>{
4 |   const text = generateText('Daniel',30);
> 5 |   expect(text).toBe('Daniel (30 years old)');
    |               ^
6 | });
7 |

at Object.<anonymous> (__test__/util.test.js:5:18)

Test Suites: 1 failed, 1 total
Tests:      1 failed, 1 total
Snapshots: 0 total
Time:       0.242 s, estimated 1 s
Ran all test suites.

Watch Usage
> Press f to run only failed tests.
> Press o to only run tests related to changed files.
> Press p to filter by a filename regex pattern.
> Press t to filter by a test name regex pattern.
> Press q to quit watch mode.
> Press i to run failing tests interactively.
> Press Enter to trigger a test run.

```


03

Agrupando teste unitário

Agrupando teste unitário

Para gerar agrupamentos de testes unitários, a seguinte palavra-chave é usada: **describe()**.

Isso será útil para organizar os testes de acordo com as funcionalidades a serem testadas.

```

EXPLORER
└─ JS-TESTING-INTRODUCTION-STARTING-SE...
   └─ __test__
      └─ JS util.test.js
         └─ .vscode
            ├── {} launch.json
            └─ dist
               ├── JS main.js
               ├── > node_modules
               ├── .gitignore
               ├── JS app.js
               ├── <> index.html
               ├── {} package-lock.json
               ├── {} package.json
               ├── # styles.css
               └─ JS util.js

__test__ > JS util.test.js > describe('Pruebas de salida de datos') callback
1  const { generateText } = require('../util.js');
2
3  describe('Pruebas de salida de datos', () => {
4      test('Salida con datos', () => {
5          const text = generateText('Daniel',30);
6          expect(text).toBe('Daniel (30 years old)');
7          const text2 = generateText('Lucas',25);
8          expect(text2).toBe('Lucas (25 years old)');
9      });
10
11      it('Salida con datos vacios', () => {
12          const text = generateText('',null);
13          expect(text).toBe(' (null years old)');
14      });
15
16      test('Salida sin datos', () => {
17          const text = generateText();
18          expect(text).toBe('undefined (undefined years old)');
19      });
20  });
21

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Ran all test suites.
PS C:\datejerina\unitTest\js-testing-introduction-starting-setup> npm test

> js-testing-introduction@1.0.0 test
> jest

PASS __test__/util.test.js
  Pruebas de salida de datos
    ✓ Salida con datos (1 ms)
    ✓ Salida con datos vacios
    ✓ Salida sin datos

Test Suites: 1 passed, 1 total
Tests: 3 passed, 3 total
Snapshots: 0 total
Time: 0.343 s, estimated 1 s
Ran all test suites.
PS C:\datejerina\unitTest\js-testing-introduction-starting-setup>
  
```

Muito obrigado!